

Pulling back error to the hidden-node parameter technology: Single-hidden-layer feedforward network without output weight

Yimin Yang, Q. M. Jonathan Wu, Guangbin Huang,
and Yaonan Wang

Abstract—According to conventional neural network theories, the feature of single-hidden-layer feedforward neural networks (SLFNs) resorts to parameters of the weighted connections and hidden nodes. SLFNs are universal approximators when at least the parameters of the networks including hidden-node parameter and output weight exist. Unlike above neural network theories, this paper indicates that in order to let SLFNs work as universal approximators, one may simply calculate the hidden node parameter only and the output weight is not needed at all. In other words, this proposed neural network architecture can be considered as a standard SLFNs with fixing output weight equal to an unit vector. Further more, this paper presents experiments which show that the proposed learning method tends to extremely reduce network output error to a very small number with only 1 hidden node. Simulation results demonstrate that the proposed method can provide several to thousands of times faster than other learning algorithm including BP, SVM/SVR and other ELM methods.

Index Terms—Bidirectional Extreme Learning Machine, Feedforward neural network, universal approximation, number of hidden nodes, learning effectiveness

I. INTRODUCTION

The widespread popularity of neural networks in many fields is mainly due to their ability to approximate complex nonlinear mappings directly from the input samples. In the past two decades, due to their universal approximation capability, feedforward neural networks (FNNs) have been extensively used in classification and regression problem[1]. According to Jaeger's estimation[2], 95% literatures are mainly on FNNs. As a specific type of FNNs, the single-hidden-layer feedforward network (SLFNs) plays an important role in practical applications[3]. For N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{in}]^T \in \mathbf{R}^n$ and $\mathbf{t}_i \in \mathbf{R}^m$, an SLFNs with L hidden nodes and activation function $h(x)$ are mathematically modeled as

$$f_L(\mathbf{x}) = \sum_{i=1}^L \beta_i h(\mathbf{a}_i \cdot \mathbf{x}_j + b_i), j = 1, \dots, N \quad (1)$$

where $h(\mathbf{a}_i, b_i, \mathbf{x})$ denotes the output of the i th hidden node with the hidden-node parameters $(\mathbf{a}_i, b_i) \in \mathbf{R}^n \times \mathbf{R}$ and $\beta_i \in \mathbf{R}$ is the output weight between the i th hidden node and the output node. $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of vector \mathbf{a}_i and \mathbf{x} in \mathbf{R}^n .

An active topic on the universal approximation capability of SLFNs is then how to determine the parameters \mathbf{a}_i, b_i , and β_i ($i = 1, \dots, L$) such that the network output $f_L(\mathbf{x})$ can approximate a given target $\mathbf{T}, \mathbf{T} = [\mathbf{t}_1, \dots, \mathbf{t}_N]$. The feature of SLFNs resorts to parameters of the output weight and hidden nodes parameters. According to conventional neural network theories, SLFNs are universal approximators when all the parameters of the networks

This work was supported by National Natural Science Foundation of China (61175075, 61301254, 61304007), and and Hunan Provincial Innovation Foundation For Postgraduate (CX2012B147).

Y. M. Yang is with the College of Electric Engineering, Guangxi University, Nanning 530004, China, and also with the Department of Electrical and Computer Engineering, University of Windsor N9B 3P4, Canada.

Q. M. Jonathan Wu is with the Department of Electrical and Computer Engineering, University of Windsor N9B 3P4, Canada.

G. B. Huang is with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798, Singapore.

Y. N. Wang are with the College of Electrical and Information Engineering, Hunan University, Changsha 410082, China.

including the hidden-node parameters (\mathbf{a}, b) and output weight β are allowed adjustable[4][5].

Unlike above neural network theories that all the parameters in networks are allowed adjustable, other researches proposed some semi-random network theories[6][7][8]. For example, Lowe [8] focus on a specific RBF network: The centers a in [8] can be randomly selected from the training data instead of tuning, but the impact factor b of RBF hidden node is not randomly selected and usually determined by users.

Unlike above semi-random network theories, in 2006, Huang *et al*[9] illustrated that iterative techniques are not required in adjusting all the parameters of SLFNs at all. Based on this idea, Huang *et al* proposed simple and efficient learning steps referred to as extreme learning machine (ELM). In [10][11][12], Huang *et al* have proved that SLFNs with randomly generated hidden node parameter can work as universal approximators by only calculating the output weights linking the hidden layer to the output nodes. Recently ELM development [13] shows that ELM unifies FNNs and SVM/LS-SVM. Compared to ELM, LS-SVM and PSVM achieve suboptimal solutions and have a higher computational cost.

Above neural network theories indicate that SLFNs can work as universal approximation at least hidden-node parameters¹ and output weight should be exist, however, in this paper we indicate that output weight do not need exist in SLFNs at all.

In [14] we proposed a learning algorithm, called bidirectional extreme learning machine (B-ELM) in which half of hidden-node parameters are not randomly selected and are calculated by pulling back the network residual error to input weight. The experimental results in [14] indicated that B-ELM tends to reduce network output error to a very small value at an extremely early learning stage. Further more, our recent experimental results indicate that in B-ELM[14], output weight play a very minion role in the network learning effectiveness. Inspired by these experimental results, in this paper, we show that SLFNs without output weight can approximate any target continuous function and classify any disjoint regions if one using pulling back error to hidden-node parameters. In particular, the following contributions have been made in this paper.

1) The learning speed of proposed learning method can be several to thousands of times faster than other learning methods including SVM, BP and other ELMs. Further more, it can provide good generalization performance and can be applied in regression and classification applications directly.

2) Different from conventional SLFNs in which the hidden node parameter and output weight should be needed, in the proposed method, we proved that SLFNs without output weight can still approximate any target continuous function and classify any disjoint regions. Thus the architecture of this single parameter neural network is extremely simpler than traditional SLFNs.

3) Different from other neural networks requiring large number of hidden nodes², experimental study shows that the proposed learning method with only one hidden node can give significant improvements on accuracy instead of maintaining a large hidden-numbers hidden layer.

¹hidden-node parameters can be generated randomly

²In [13], Huang *et al* indicate "The generalization performance of ELM is not sensitive to the dimensionality L of the feature space (the number of hidden nodes) as long as L is set large enough (e.g., $L > 1000$ for all the real-world cases tested in our simulations)."

II. PRELIMINARIES AND NOTATION

A. Notations and Definitions

The sets of real, integer, positive real and positive integer numbers are denoted by $\mathbf{R}, \mathbf{Z}, \mathbf{R}^+$ and \mathbf{Z}^+ , respectively. Similar to [1], let $F^2(X)$ be a space of functions f on a compact subset X in the n -dimensional Euclidean space \mathbf{R}^n such that $|f|^2$ are integrable, that is, $\int_X |f(x)|^2 dx < \infty$. Let $F^2(\mathbf{R}^n)$ be denoted by F^2 . For $u, v \in F^2(X)$, the inner product $\langle u, v \rangle$ is defined by

$$\langle u, v \rangle = \int_X u(\mathbf{x}) \overline{v(\mathbf{x})} d\mathbf{x} \quad (2)$$

The norm in $F^2(X)$ space will be denoted as $\|\cdot\|$. L denotes the number of hidden nodes. For N training samples, $\mathbf{x}, \mathbf{x} \in \mathbf{R}^{N \times n}$ denotes the input matrix of network, $\mathbf{T} \in \mathbf{R}^{N \times m}$ denotes the desire output matrix of network. $\mathbf{H} \in \mathbf{R}^{N \times m}$ is called the hidden layer output matrix of the SLFNs; the i th column of \mathbf{H} (\mathbf{H}_i) is the i th hidden node output with respect to inputs. The hidden layer output matrix \mathbf{H}_i is said to be randomly generated function sequence \mathbf{H}_i^e if the corresponding hidden-node parameters (\mathbf{a}_i, b_i) are randomly generated. $\mathbf{e}_n, \mathbf{e}_n \in \mathbf{R}^{N \times m}$ denotes the residual error function for the current network f_n with n hidden nodes. \mathbf{I} is unit matrix and $\mathbf{I} \in \mathbf{R}^{m \times m}$.

III. BIDIRECTIONAL ELM FOR REGRESSION PROBLEM

Theorem 1: [14] Given N training samples $\{(\mathbf{x}_i, t_i)\}_{i=1}^N \in \mathbf{R}^n \times \mathbf{R}$ come from the same continuous function, given the sigmoid or sine activation function $h: \mathbf{R} \rightarrow \mathbf{R}$; Given an error feedback function sequence $\mathbf{H}_{2n}^e(\mathbf{x}, \mathbf{a}, b)$ by

$$\mathbf{H}_{2n}^e = \mathbf{e}_{2n-1} \cdot (\beta_{2n-1})^{-1} \quad (3)$$

If activation function h is sin/cos, given a normalized function $u: \mathbf{R} \rightarrow [0, 1]$; If activation function h is sigmoid, given a normalized function $u: \mathbf{R} \rightarrow (0, 1)$. Then for any continuous target function f , randomly generated function sequence \mathbf{H}_{2n+1}^r , $\lim_{n \rightarrow \infty} \|f - (\mathbf{H}_1^r \cdot \beta_1 + \hat{\mathbf{H}}_2^e(\hat{\mathbf{a}}_2, \hat{b}_2) \cdot \beta_2 + \dots + \mathbf{H}_{2n-1}^r \cdot \beta_{2n-1} + \hat{\mathbf{H}}_{2n}^e(\hat{\mathbf{a}}_{2n}, \hat{b}_{2n}) \cdot \beta_{2n})\| = 0$ hold with probability one if

$$\hat{\mathbf{a}}_{2n} = h^{-1}(u(\mathbf{H}_{2n}^e)) \cdot \mathbf{x}^{-1}, \quad \hat{\mathbf{a}}_{2n} \in \mathbf{R}^n \quad (4)$$

$$\hat{b}_{2n} = \sqrt{\text{mse}(h^{-1}(u(\mathbf{H}_{2n}^e)) - \hat{\mathbf{a}}_{2n} \cdot \mathbf{x})}, \quad \hat{b}_{2n} \in \mathbf{R} \quad (5)$$

$$\beta_{2n} = \frac{\mathbf{e}_{2n-1} \cdot \mathbf{H}_{2n}^e}{\mathbf{H}_{2n}^e \cdot (\mathbf{H}_{2n}^e)^T}, \quad \beta_{2n} \in \mathbf{R} \quad (6)$$

$$\beta_{2n+1} = \frac{\mathbf{e}_{2n} \cdot \mathbf{H}_{2n+1}^r}{\mathbf{H}_{2n+1}^r \cdot (\mathbf{H}_{2n+1}^r)^T}, \quad \beta_{2n+1} \in \mathbf{R} \quad (7)$$

where h^{-1} and u^{-1} represent its reverse function, respectively. if h is sine activation function, $h^{-1}(\cdot) = \arcsin(\cdot)$; if h is sigmoid activation function, $h^{-1}(\cdot) = -\log(\frac{1}{u(\cdot)} - 1)$.

Remark 1: Compared with B-ELM, In the proposed method, we only make two changes. The first one is we set $\beta_1 = \dots = \beta_{n-1} = \dots = \mathbf{I}$. The second one is the pseudoinverse of input data \mathbf{x}^{-1} has been changed as $\mathbf{x}^{-1} = \mathbf{x}^T (\mathbf{I} + \mathbf{x}\mathbf{x}^T)^{-1}$ based on the ridge regression theory. Although very small changes are made, the experimental results show that by using this proposed learning method, one hidden-node SLFNs without output weight (output weight β equal to unit matrix) can achieve similar generalization performance as other standard SLFNs with hundreds of hidden nodes. Further more, different from B-ELM [14] which only work for regression problem, the proposed method can be applied in regression and multi-classification applications.

IV. SLFNs WITHOUT OUTPUT WEIGHT

Basic idea 1: our recent experimental results indicate that in B-ELM[14], output weight play a very minion role in the network learning effectiveness. Inspired by these experimental results, in this proposed method, we directly set output weight equal to unit matrix.

Theorem 2: Given N training samples $\{(\mathbf{x}_i, t_i)\}_{i=1}^N \in \mathbf{R}^n \times \mathbf{R}^m$ come from the same continuous function, given an SLFNs with any bounded nonconstant piecewise continuous function $\mathbf{H}: \mathbf{R} \rightarrow \mathbf{R}$ for additive nodes or sine nodes, for any continues target function f , obtained error feedback function sequence $\mathbf{H}_n^e, n \in \mathbf{Z}$, $\lim_{n \rightarrow \infty} \|f - (f_{n-1} + \mathbf{H}_n^e \cdot \beta_{n-1})\| = 0$ holds with probability one if

$$\mathbf{H}_n^e = \mathbf{e}_{n-1} \cdot (\beta_{n-1})^{-1} \quad (8)$$

$$\beta_n = \beta_{n-1} = \mathbf{I}, \beta_n \in \mathbf{R}^{m \times m} \quad (9)$$

Proof: The validity of this theorem is obvious because $\beta_{n-1} = \mathbf{I}$ and $(\beta_{n-1})^{-1} = \mathbf{I}$, \mathbf{H}_n^e equal to \mathbf{e}_{n-1} . And we can get $\|\mathbf{e}_n\| = 0$. ■

Remark 2: When $\mathbf{H}_n^e = \mathbf{e}_{n-1} = \mathbf{T}$, it is easy to notice that the proposed method can reduce the network output error to 0. Thus the learning problem has been converted into finding optimal hidden node parameter (\mathbf{a}, b) which lead to $\mathbf{H}(\mathbf{a}, b, \mathbf{x}) \rightarrow \mathbf{T}$.

Basic idea 2: For fixed output weight β equal to unit matrix or vector ($\beta \in \mathbf{R}^{m \times m}$), seen from equation (8)-(9), to train an SLFN is simply equivalent to finding a least-square solution \mathbf{a}^{-1} of the linear system $\mathbf{H}(\mathbf{a}, \mathbf{x}) = \mathbf{T}$. If activation function can be invertible, to train an SLFN is simply equivalent to pulling back residual error to input weight. For example, for N arbitrary distinct samples $\{\mathbf{x}, \mathbf{T}\}$, $\mathbf{x} \in \mathbf{R}^{N \times n}, \mathbf{T} \in \mathbf{R}^{N \times m}, \mathbf{T} \in [0, 1]$, If activation function is sine function, to train an SLFN is simply equivalent to finding a least-square solution $\hat{\mathbf{a}}$ of the linear system $\mathbf{a} \cdot \mathbf{x} = \arcsin(\mathbf{T})$:

$$\|\mathbf{H}(\hat{\mathbf{a}}_1, \dots, \hat{\mathbf{a}}_n, \mathbf{x}) - \mathbf{T}\| = \min_{\mathbf{a}} \|\mathbf{H}(\mathbf{a}_1, \dots, \mathbf{a}_n, \mathbf{x}) - \mathbf{T}\| \quad (10)$$

According to [16], the smallest norm least-squares solution of the above linear system is $\hat{\mathbf{a}}_n = \arcsin(\mathbf{e}_{n-1}) \cdot \mathbf{x}^{-1}$. Based on this idea, we give the following theorem.

Lemma 1: [1] Given a bounded nonconstant piecewise continuous function $H: \mathbf{R} \rightarrow \mathbf{R}$, we have

$$\lim_{(\mathbf{a}, b) \rightarrow (\mathbf{a}_0, b_0)} \|\mathbf{H}(\mathbf{a} \cdot \mathbf{x} + b) - \mathbf{H}(\mathbf{a}_0 \cdot \mathbf{x} + b_0)\| = 0 \quad (11)$$

Theorem 3: Given N arbitrary distinct samples $\{\mathbf{x}, \mathbf{T}\}, \mathbf{x} \in \mathbf{R}^{N \times n}, \mathbf{T} \in \mathbf{R}^{N \times m}$, given the sigmoid or sine activation function h , for any continuous desire output \mathbf{T} , there exist $\lim_{n \rightarrow \infty} \|\mathbf{T} - (\hat{\mathbf{H}}_1(\hat{\mathbf{a}}_1, \hat{b}_1, \mathbf{x})\beta_1 + \dots + \hat{\mathbf{H}}_n(\hat{\mathbf{a}}_n, \hat{b}_n, \mathbf{x})\beta_n)\| = 0$ hold with probability one if

$$\mathbf{H}_n^e = \mathbf{e}_{n-1} \quad (12)$$

$$\hat{\mathbf{a}}_n = h^{-1}(u(\mathbf{H}_n^e)) \cdot \mathbf{x}^{-1}, \quad \hat{\mathbf{a}}_n \in \mathbf{R}^{n \times m}$$

$$\hat{b}_n = \sqrt{\text{mse}(h^{-1}(u(\mathbf{H}_n^e)) - \hat{\mathbf{a}}_n \cdot \mathbf{x})}, \quad \hat{b}_n \in \mathbf{R}^m$$

$$\beta_1 = \beta_2 = \dots = \beta_n = \mathbf{I} \quad (13)$$

where if activation function h is sin/cos, given a normalized function $u: \mathbf{R} \rightarrow [0, 1]$; If activation function h is sigmoid, given a normalized function $u: \mathbf{R} \rightarrow (0, 1)$. h^{-1} and u^{-1} represent its reverse function, respectively. If h is sine activation function, $h^{-1}(\cdot) = \arcsin(u(\cdot))$; if h is sigmoid activation function, $h^{-1}(\cdot) = -\log(\frac{1}{u(\cdot)} - 1)$, $\mathbf{x}^{-1} = \mathbf{x}^T (\mathbf{I} + \mathbf{x}\mathbf{x}^T)^{-1}$.

Proof: For an activation function $h(x): \mathbf{R} \rightarrow \mathbf{R}$, \mathbf{H}_n^e is given by

$$\mathbf{H}_n^e = h(\lambda_n) \quad (14)$$

In order to let $\lambda_{2n} \in \mathbf{R}^m$, here we give a normalized function $u(\cdot)$: $u(\mathbf{H}) \in [0, 1]$ if activation function is sin/cos; $u(\mathbf{H}) \in (0, 1)$ if activation function is sigmoid. Then for sine hidden node

$$\lambda_{2n} = h^{-1}(u(\mathbf{H}_n^e)) = (\arcsin(u(\mathbf{H}_n^e))) \quad (15)$$

For sigmoid hidden node

$$\lambda_n = h^{-1}(u(\mathbf{H}_n^e)) = -\log\left(\frac{1}{u(\mathbf{H}_n^e)} - 1\right) \quad (16)$$

let $\lambda_n = \mathbf{a}_n \cdot \mathbf{x}$, for sine activation function, we have

$$\hat{\mathbf{a}}_n = h^{-1}(u(\mathbf{H}_n^e)) \cdot \mathbf{x}^{-1} = \arcsin(u(\mathbf{H}_n^e)) \cdot \mathbf{x}^{-1} \quad (17)$$

For sigmoid activation function, we have

$$\hat{\mathbf{a}}_n = h^{-1}(u(\mathbf{H}_n^e)) \cdot \mathbf{x}^{-1} = -\log\left(\frac{1}{u(\mathbf{H}_n^e)} - 1\right) \cdot \mathbf{x}^{-1} \quad (18)$$

where \mathbf{x}^{-1} is the Moore-Penrose generalized inverse of the given set of training examples[15]. Similar to [16], we have 1: $\hat{\mathbf{a}}_n = \arcsin(u(\mathbf{H}_n^e)) \cdot \mathbf{x}^{-1}$ is one of the least-squares solutions of a general linear system $\mathbf{a}_n \cdot \mathbf{x} = \lambda_n$, meaning that the smallest error can be reached by this solution:

$$\|\hat{\mathbf{a}}_n \cdot \mathbf{x} - \lambda_n\| = \|\hat{\mathbf{a}}_n \mathbf{x}^{-1} \mathbf{x} - \lambda_n\| = \min_{\mathbf{a}_n} \|\mathbf{a}_n \cdot \mathbf{x} - \arcsin(u(\mathbf{H}_n^e))\| \quad (19)$$

2: the special solution $\hat{\mathbf{a}}_n = h^{-1}(u(\mathbf{H}_n^e)) \cdot \mathbf{x}^{-1}$ has the smallest norm among all the least-squares solutions of $\mathbf{a}_n \cdot \mathbf{x} = \lambda_n$, which is guarantee that $\mathbf{a}_n \in [-1, 1]$. Although the smallest error can be reached by equation (17)-(18), we still can reduce its error by adding bias b_n . For sine activation function:

$$\begin{aligned} \hat{b}_n &= \sqrt{\text{mse}(h^{-1}(u(\mathbf{H}_n^e)) - \hat{\mathbf{a}}_n \cdot \mathbf{x})} \\ &= \sqrt{\text{mse}(\arcsin(u(\mathbf{H}_n^e)) - \hat{\mathbf{a}}_n \cdot \mathbf{x})} \end{aligned} \quad (20)$$

For sigmoid activation function

$$\begin{aligned} \hat{b}_n &= \sqrt{\text{mse}(h^{-1}(u(\mathbf{H}_n^e)) - \hat{\mathbf{a}}_n \cdot \mathbf{x})} \\ &= \sqrt{\text{mse}((- \log(1/u(\mathbf{H}_n^e)) - 1) - \hat{\mathbf{a}}_n \cdot \mathbf{x})} \end{aligned} \quad (21)$$

According to 19 and Lemma 1, we have

$$\begin{aligned} \min_{\mathbf{a}_n} \|u^{-1}(h(\mathbf{a}_n \cdot \mathbf{x})) - u^{-1}(h(\lambda_n))\| \\ = \|u^{-1}(h(\hat{\mathbf{a}}_n \cdot \mathbf{x})) - u^{-1}(h(\lambda_n))\| \\ > \|u^{-1}(h(\hat{\mathbf{a}}_n \cdot \mathbf{x} + \hat{b}_n)) - u^{-1}(h(\lambda_n))\| = \|\sigma\| \end{aligned} \quad (22)$$

We consider the residual error as

$$\begin{aligned} \Delta &= \|e_{n-1}\|^2 - \|e_{n-1} - \mathbf{H}_n^e \cdot \beta_n\|^2 \\ &= 2\beta_n \langle e_{n-1}, \mathbf{H}_n^e \rangle - \|\mathbf{H}_n^e\|^2 \cdot \beta_n^2 \\ &= \|\mathbf{H}_n^e\|^2 \left(\frac{2\beta_n \langle e_{n-1}, \mathbf{H}_n^e \rangle}{\|\mathbf{H}_n^e\|^2} - \beta_n^2 \right) \end{aligned} \quad (23)$$

Let

$$\begin{aligned} \hat{\mathbf{H}}_n^e &= u^{-1}(h(\hat{\mathbf{a}}_n \cdot \mathbf{x} + \hat{b}_n)) \\ &= \frac{e_{n-1} - \sigma}{\beta_n} \\ &= \frac{\hat{e}_{n-1}}{\beta_{n-1}} \end{aligned} \quad (24)$$

Because $\beta_n = \beta_{n-1} = \mathbf{I}$, we have equation (23) ≥ 0 is still valid for

$$\begin{aligned} \Delta &= \|\hat{\mathbf{H}}_n^e\|^2 \left(\frac{2\|\beta_n\| \langle \hat{e}_{n-1}, \frac{\hat{e}_{n-1}}{\beta_{n-1}} \rangle}{\|\frac{\hat{e}_{n-1}}{\beta_{n-1}}\|^2} - \|\beta_n\|^2 \right) \\ &= \|\hat{\mathbf{H}}_n^e\|^2 \left(\frac{2\|\hat{e}_{n-1}\|^2}{\|\frac{\hat{e}_{n-1}}{\beta_{n-1}}\|^2} - \beta_n^2 \right) \\ &= \|\hat{\mathbf{H}}_n^e\|^2 \beta_n^2 \geq 0 \end{aligned} \quad (25)$$

Now based on equation 25, we have $\|\mathbf{e}_{n-1}\| \geq \|\mathbf{e}_n\|$, so the sequence $\|\mathbf{e}_n\|$ is decreasing and bounded below by zero and the sequence $\|\mathbf{e}_n\|$ converges. ■

Remark 3: According to Theorem 2-3, for N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$ where $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{iN}]^T \in \mathbf{R}^n$ and $\mathbf{t}_i \in \mathbf{R}^m$, the proposed network with L hidden nodes and activation function $h(x)$ are mathematically modeled as

$$f_L(\mathbf{x}) = \sum_{i=1}^L u^{-1}(h(\mathbf{a}_i \cdot \mathbf{x}_j + b_i)), j = 1, \dots, N \quad (26)$$

where u is a normalized function, $\mathbf{a}_i \in \mathbf{R}^{n \times m}$, $b_i \in \mathbf{R}^m$. Here, the proposed the proposed method for SLFN can be summarized in Algorithm 1.

Algorithm 1 the proposed method algorithm

Initialization: Given a training set $\{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^N \subset \mathbf{R}^n \times \mathbf{R}^m$, the hidden-node output function $\mathbf{H}(\mathbf{a}, b, \mathbf{x})$, continuous target function f , set number of hidden nodes $L = 1$, $\mathbf{e} = \mathbf{T}$.

Learning step:

while $L < L_{max}$ **do**

 Increase by one the number of hidden nodes $L; L = L + 1$;

 Step 1) set $\mathbf{H}_L^e = \mathbf{e}$;

 Step 2) calculate the input weight \mathbf{a}_L , bias b_L based on equation 12;

 Step 3) calculate \mathbf{e} after adding the new hidden node L :

$\mathbf{e} = \mathbf{e} - u^{-1}(h(\mathbf{a} \cdot \mathbf{x} + b))$

end while

Remark 4: Different from other neural network learning methods in which output weight parameter should be adjusted, in the proposed method, the output weight of SLFNs can be equal to unit matrix and thus the proposed neural network does not need output weight at all. Thus the architecture and computational cost of this proposed method are much smaller than other traditional SLFNs.

Remark 5: Subsection V.C presents experiments which show that the proposed method with only one hidden node can give better generalization performance than the proposed network with $L(L > 1)$ hidden node. Based on this experimental results, for N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$ where $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{iN}]^T \in \mathbf{R}^n$ and $\mathbf{t}_i \in \mathbf{R}^m$, the proposed network is mathematically modeled as

$$f_L(\mathbf{x}) = u^{-1}(h(\mathbf{a}_1 \cdot \mathbf{x}_j + b_1)), j = 1, \dots, N \quad (27)$$

where u is a normalized function, $\mathbf{a}_1 \in \mathbf{R}^{n \times m}$, $b_1 \in \mathbf{R}^m$. Thus algorithm 1 can be modified as algorithm 2.

Algorithm 2 the proposed method algorithm

Initialization: Given a training set $\{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^N \subset \mathbf{R}^n \times \mathbf{R}^m$, the hidden-node output function $\mathbf{H}(\mathbf{a}, b, \mathbf{x})$, continuous target function f , set number of hidden nodes $L = 1$.

Learning step:

Step 1) set $\mathbf{H}_1^e = \mathbf{T}$;

Step 2) calculate the input weight \mathbf{a}_1 , bias b_1 based on equation 12;

V. EXPERIMENTAL VERIFICATION

To examine the performance of our proposed algorithm (B-ELM), in this section, we test them on some benchmark regression and classification problems. Neural networks are tested in SVR, SVM, BP, EM-ELM, I-ELM, EI-ELM, B-ELM, ELM and proposed the proposed method.

TABLE I
SPECIFICATION OF REGRESSION PROBLEMS

Datasets	#Attri	#Train	# Test
Auto MPG	8	200	192
Machine CPU	6	100	109
Fried	11	20768	20000
Wine Quality	12	2898	2000
Puma	9	4500	3692
California Housing	8	16000	4000
House 8L	9	16000	6784
Parkinsons motor	26	4000	1875
Parkinsons total	26	4000	1875
Puma	9	6000	2192
Delta elevators	6	6000	3000
Abalone	9	3000	1477

TABLE II
SPECIFICATION OF SMALL/MEDIUM-SIZED CLASSIFICATION PROBLEMS

Datasets	#Feature	#Train	# Test
A9a	123	32561	16281
colon-cancer	2000	40	22
USPS	256	7291	2007
Sonar	60	150	58
Hill Valley	101	606	606
Protein	357	17766	6621

TABLE III
SPECIFICATION OF LARGE-SIZED CLASSIFICATION PROBLEMS

Datasets	#Feature	#Train	# Test
Covtype.binary	54	300000	280000
Mushrooms	112	4000	4122
Gisette	5000	6000	1000
Leukemia	7129	38	34
Duke	7129	29	15
Connect-4	126	50000	17557
Mnist	780	40000	30000
DNA	180	1046	1186
w3a	300	4912	44837

A. Benchmark Data Sets

In order to extensively verify the performance of different algorithms, wide type of data sets have been tested in our simulations, which are of *small size*, *medium dimensions*, *large size*, and/or *high dimensions*. These data sets include 12 regression problems and 15 classification problems. Most of the data sets are taken from UCI Machine Learning Repository³ and LIBSVM DATA SETS⁴.

Regression Data Sets: The 12 regression data sets(cf.Table I) can be classified into two groups of data:

- 1) data sets with relatively small size and low dimensions, e.g., Auto MPG, Machine CPU, Puma, Wine, Abalone;
- 2) data sets with relatively medium size and low dimensions, e.g., Delta, Fried, California Housing, Parkinsons;

Classification Data Sets: The 15 classification data sets(cf.Table II and Table III) can be classified into three groups of data:

- 1) data sets with relative medium size and medium dimensions, e.g., Sonar, Hill Valley, Wa3, DNA, Mushrooms, A9a, USPS;
- 2) data sets with relative small size and high dimensions, e.g., Colon-cancer, Leukemia, Duke;
- 3) data sets with relative large size and high dimensions, e.g., Protein, Covtype.binary, Gisette, Mnist, Connect-4;

In these data sets, the input data are normalized into $[-1, 1]$ while the output data for regression are normalized into the range $[0, 1]$. All data sets have been preprocessed in the same way (held-out method). Ten different random permutations of the whole data set are taken without replacement, and some(see in tables) are used to create the training set and the remaining is used for the test set. The average results are obtained over 50 trials for all problems.

B. Simulation Environment Settings

The simulations of different algorithms on the data sets which are shown in Table I and Table II are carried out in Matlab 2009a environment running on the same Windows 7 machine with at 2 GB of memory and an i5-430 (2.33G) processor. The codes used for SVM and SVR are downloaded from LIBSVM⁵, The codes used for B-ELM, ELM and I-ELM are downloaded from ELM⁶.

For SVM and SVR, in order to achieve good generalization performance, the cost parameter C and kernel parameter γ of SVM and SVR need to be chosen appropriately. We have tried a wide range of C and γ . For each data set, similar to [17], we have used 30 different value of C and γ , resulting in a total of 900 pairs of (C, γ) . The 30 different value of C and γ are $\{2^{-15}, 2^{-14}, \dots, 2^{14}, 2^{15}\}$. Average results of 50 trials of simulations with each combination of (C, γ) are obtained and the best performance obtained by SVM/SVR are shown in this paper.

For BP, the number of hidden nodes are gradually increased by an interval of 5 and the nearly optimal number of nodes for BP are then selected based on cross-validation method. Average results of 50 trails of simulations for each fixed size of SLFN are obtained and finally the best performance obtained by BP are shown in this paper as well.

Simulations on large data sets(cf.Table III) are carried out in a high-performance computer with Intel Xeon E3-1230 v2 processor (3.2G) and 16-GB memory.

C. Generalization performance comparison of ELM methods with different hidden nodes

The aim of this subsection is to show that the proposed method with only one hidden node generally achieves better generalization performance than other learning methods. And it is also to show that the proposed method with one hidden node achieves the best performance than the proposed method with $L, L > 1$ one hidden node. In this subsection, I-ELM, ELM, EI-ELM and the proposed method are compared in one regression problem and three classification problems: Fried, DNA, USPS and Mushroom. In these cases, all the algorithms increase the hidden nodes one by one. More importantly, we find that the testing accuracy obtained by proposed method is reduced to a very high value when only one hidden node is used. And the testing accuracy obtained by proposed method is not increased but is reduced when hidden node added one by one. This means the proposed method only need to calculates one-hidden-node parameter(\mathbf{a}_1, b_1) once and then SLFNs without output weight can achieve similar generalization performance as other learning method with hundreds of hidden nodes. Thus in the following experiments, the number of hidden node equal to one in the proposed method.

³<http://archive.ics.uci.edu/ml/datasets.html>

⁴<http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>

⁵<http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>

⁶http://www.ntu.edu.sg/home/egbhuang/elm_codes.html

TABLE IV
PERFORMANCE COMPARISON (MEAN-MEAN TESTING RMSE; TIME-TRAINING TIME)

Datasets	I-ELM (200 nodes)		B-ELM (200 nodes)		EI-ELM (200 nodes, $p = 50$)		the proposed method (1 nodes)	
	Mean	time(s)	Mean	time(s)	Mean	time(s)	Mean	time(s)
House 8L	0.0946	1.1872	<u>0.0818</u>	3.8821	0.0850	10.7691	<u>0.0819</u>	0.0020
Auto MPG	0.1000	0.2025	0.0920	0.3732	0.0918	1.3004	0.0996	< 0.0001
Machine CPU	0.0591	0.1909	0.0554	0.3469	0.0551	1.2633	0.0489	< 0.0001
Fried	0.1135	0.8327	0.0857	5.5063	0.0856	7.4016	0.0834	0.0051
Delta ailerons	0.0538	0.4680	<u>0.0431</u>	1.3946	0.0417	3.5478	<u>0.0453</u>	< 0.0001
PD motor	0.2318	0.4639	0.2241	4.7680	0.2251	3.9016	0.2210	0.0037
PD total	0.2178	0.4678	<u>0.2137</u>	4.9278	0.2124	3.7854	<u>0.2136</u>	0.0023
Puma	0.1860	0.5070	0.1832	2.1846	0.1830	4.2161	0.1808	0.0012
Delta ele	0.1223	0.5313	<u>0.1156</u>	1.6206	<u>0.1155</u>	4.0240	0.1174	< 0.0001
Abalone	0.0938	0.3398	<u>0.0808</u>	1.2549	<u>0.0848</u>	2.6676	<u>0.0828</u>	0.0017
Wine	0.1360	0.3516	<u>0.1264</u>	1.7098	0.1266	2.7126	<u>0.1250</u>	0.0031
California house	0.1801	1.1482	0.1450	7.2625	0.1505	12.0832	<u>0.1420</u>	0.0078

TABLE V
PERFORMANCE COMPARISON (MEAN-MEAN TESTING RMSE; TIME-TRAINING TIME)

Datasets	EM-ELM (200 nodes)		ELM (200 nodes)		the proposed method (1 nodes)	
	Mean	time(s)	Mean	time(s)	Mean	time(s)
House 8L	0.0663	7.0388	0.0718	0.8369	0.0819	0.0020
Auto MPG	<u>0.0968</u>	0.0075	<u>0.0976</u>	0.0156	<u>0.0996</u>	< 0.0001
Machine CPU	0.0521	0.1385	0.0513	0.0069	0.0489	< 0.0001
Fried	<u>0.0618</u>	18.0290	<u>0.0619</u>	1.3135	0.0834	0.0051
Delta ailerons	<u>0.0421</u>	0.1342	<u>0.0431</u>	0.0616	<u>0.0453</u>	< 0.0001
PD motor	<u>0.2196</u>	0.7394	<u>0.2190</u>	0.2730	<u>0.2203</u>	0.0037
PD total	<u>0.2094</u>	0.5944	<u>0.2076</u>	0.2838	0.2136	0.0023
Puma	0.1478	4.8392	0.1602	0.3728	0.1808	0.0012
Abalone	<u>0.0817</u>	0.1638	<u>0.0824</u>	0.0761	<u>0.0828</u>	0.0017
Wine	<u>0.1216</u>	0.3806	<u>0.1229</u>	0.1950	<u>0.1250</u>	0.0031
California house	0.1302	3.5574	0.1354	0.9753	0.1420	0.0078

TABLE VII
PERFORMANCE COMPARISON (MEAN-MEAN TESTING RMSE; TIME-TRAINING TIME)

Datasets	Eplison-SVR		BP		the proposed method (1 nodes)	
	Mean	time(s)	Mean	time(s)	Mean	time(s)
House 8L	<u>0.0799</u>	53.6531	<u>0.0790</u>	27.8462	0.0819	0.0020
Auto MPG	<u>0.0985</u>	0.0234	<u>0.0953</u>	1.6034	<u>0.0996</u>	< 0.0001
Machine CPU	0.0727	0.0187	0.0843	0.7129	0.0489	< 0.0001
Fried	0.0829	197.9534	0.0591	81.8774	0.0834	0.0051
Delta ailerons	0.0402	6.8718	0.0415	12.6735	0.0453	< 0.0001
California house	0.1529	35.2250	<u>0.1435</u>	54.3081	0.1420	0.0078
PD total	0.2082	7.2540	0.2120	12.6438	0.2136	0.0023

TABLE VIII
PERFORMANCE COMPARISON (MEAN-MEAN TESTING RMSE; TIME-TRAINING TIME)

Datasets	SVM		ELM		#node	the proposed method (1 nodes)	
	Mean	time(s)	Mean	time(s)		Mean	time(s)
Covtype.binary	74.84%	413.5275	<u>77.27%</u>	36.5947	500	<u>76.55%</u>	1.2043
Mushrooms	86.90%	38.6247	46.97%	0.9126	500	88.84%	0.0047
Gisette	77.68%	309.3968	88.69%	6.4093	500	94.10%	48.2027
Leukemia	82.58%	2.3914	76.47%	9.0340	5000	85.29%	20.9915
W3a	<u>97.18%</u>	4.5552	<u>97.25%</u>	0.9095	500	<u>98.17%</u>	0.1872
Duke	86.36%	0.0156	79.27%	7.8437	5000	92.67%	20.0352
Connect-4	<u>66.01%</u>	569.6221	<u>76.55%</u>	7.3757	500	<u>75.40%</u>	0.7597
Mnist	70.85%	478.4707	91.60%	8.1651	500	84.20%	8.8858
DNA	<u>93.70%</u>	0.4680	84.94%	0.2122	500	<u>92.41%</u>	0.0187

D. Real-world regression problems

The experimental results between proposed the proposed method and some other incremental ELMs (B-ELM, I-ELM, and EI-ELM) are given in Table IV-Table V. In these tables, the close

results obtained by different algorithms are underlined and the apparent better results are shown in boldface. All the incremental ELMs (I-ELM, B-ELM, EI-ELM) increase the hidden nodes one by one till nodes-numbers equal to 200, while for fixed ELMs (ELM, EM-ELM), 200-hidden-nodes are used. It can be seen that the

TABLE IX
PERFORMANCE COMPARISON (MEAN-MEAN TESTING RMSE; TIME-TRAINING TIME)

Datasets	SVM		ELM		#node	the proposed method (1 nodes)	
	Mean	time(s)	Mean	time(s)		Mean	time(s)
A9a	77.39%	295.0603	<u>85.10%</u>	4.5871	500	<u>85.57%</u>	0.5714
Colon	76.67%	10.0156	80.67%	11.6283	5000	85.06%	0.9719
USPS	<u>94.65%</u>	146.4942	<u>93.54%</u>	2.0639	500	88.86%	0.4898
Sonar	86.29%	0.0172	80.86%	0.0686	500	75.69%	<0.0001
Hill Valley	58.67%	0.1295	64.31	0.1647	500	67.61%	0.0047
Protein	51.18%	253.5796	67.09%	5.0919	500	68.76%	1.9953

TABLE VI
PERFORMANCE COMPARISON (MEAN-MEAN TESTING RMSE; TIME-TRAINING TIME)

Datasets	ELM	(1 nodes)	the proposed method (1 nodes)	
	Mean	time(s)	Mean	time(s)
House 8L	0.1083	0.0009	0.0819	0.0020
Auto MPG	0.2126	< 0.0001	0.0996	<0.0001
Machine CPU	0.1331	<0.0001	0.0489	<0.0001
Fried	0.2207	0.0031	0.0834	0.0051
Delta ailerons	0.0864	<0.0001	0.0453	<0.0001
PD motor	0.2620	0.0020	0.2210	0.0037
PD total	0.2548	0.0007	0.2136	0.0023
Puma	0.2856	<u>0.0012</u>	0.1808	0.0012
Delta ele	0.1454	<0.0001	0.1174	<0.0001
Abalone	0.1363	0.0007	0.0828	0.0017
Wine	0.1750	0.0006	0.1250	0.0031
California house	0.2496	0.0027	0.1420	0.0078

proposed method can always achieve similar performance as other ELMs with much higher learning speed. In Table IV, for Machine CPU problem, the the proposed method runs 1900 times, 3400 times, 12000 times faster than the I-ELM, B-ELM and EI-ELM, respectively. For Abalone problem, the proposed method runs 200 times, 700 times, 1600 times faster than I-ELM, B-ELM and EI-ELM, respectively. In Table V, for Wine problem, the the proposed method runs 120 times and 60 times faster than EM-ELM and ELM, respectively. and the testing RMSE of EI-ELM is 2 times larger than the testing RMSE of B-ELM. The B-ELM runs 1.5 times faster than the I-ELM and the testing RMSE for the obtained I-ELM is 5 times larger than the testing RMSE for B-ELM.

If only 1-hidden-node being used, those ELM methods such as I-ELM, ELM, EM-ELM and B-ELM can be considered as the same learning method (ELM[13]). Thus in Table VI, we carried out performance comparisons between the proposed method and 1-hidden-node ELM. As observed from Table VI, the average testing RMSE obtained by the proposed method are much better than the ELM. For California house and Delta ailerons problem, the testing RMSE obtained by ELM runs 2 times larger than that of the proposed method. In real applications, SLFNs with only 1 hidden nodes is extremely small network structure, meaning that after trained this small size network may response to new external unknown stimuli much faster and much more accurate than other ELM algorithms in real deployment.

E. Real-world classification problems

In order to indicate the advantage of the B-ELM on classification performance, the testing accuracy between the proposed the proposed method and other algorithms has also been conducted. Table VIII and IX display the performance comparison of SVM, ELM and the proposed method. In these tables, the close results obtained by different algorithms are underlined and the apparent

better results are shown in boldface. As seen from those simulation results given in these tables, the proposed method can always achieve comparable performance as SVM and ELM with much faster learning speed. Take Covtype.binary (large number of training samples with medium input dimensions) and Gisette (medium number of training samples with high input dimensions).

1) For Covtype.binary data set, the proposed method runs 1403 times and 35 times faster than ELM and SVM, respectively.

2) For Gisette data set, the proposed method runs 341 times and 1.7 times faster than ELM and SVM, respectively.

Huang *et al.*[1][18][16][13] have systematically investigated the performance of ELM, SVM/SVR and BP for most data sets tested in this work. It is found that ELM obtain similar generalization performance as SVM/SVR but in much simpler and faster way. Similar to those above works, our testing results(cf. Table VII-IX) shows that the proposed the proposed method always provide comparable performance as SVM/SVR and BP with much faster learning speed.

On the other hand, the proposed method requires none human intervention than SVM, BP and other ELM methods. Different from SVM which is sensitive to the combinations of parameters (C, γ), or from other ELM methods in which parameter C needs to be specified by users, the proposed method have none specified parameter and is ease of use in the respective implementations.

VI. CONCLUSION

Unlike other SLFN learning methods, in our new approach, one may simply calculate the hidden node parameter once and the output weight is not need at all. And it has been rigorously proved that the proposed method can greatly enhance the learning effectiveness, reduce the computation cost, and eventually further increase the learning speed. The simulation results on sigmoid type of hidden nodes show that compared to other learning methods including SVM/SVR, BP and ELMs, the new approach can significantly reduce the NN training time several to thousands of times and can applied in regression and classification problems. Thus this method can be used efficiently in many applications.

However, we find an interesting phenomenon which we are not able to prove in this method, which should be worth pointing out. Experimental results show that this proposed learning method with one hidden node can achieve better generalization performance than the same method with $L, L > 1$ hidden nodes. This phenomenon of this proposed method bring about many advantages, but if researchers can find the nature of this phenomenon, it can have far reaching consequences on the generalization ability of neural network.

REFERENCES

- [1] Guang-Bin Huang, Lei Chen, and Chee-Kheong Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, 2006. [1, 2, 6]

- [2] Herbert Jaeger. A tutorial on training recurrent neural networks , covering BPPT , RTRL , EKF and the " echo state network " approach. 2002:1–46, 2005. [1]
- [3] G. B. Huang, P. Saratchandran, and N. Sundararajan. An efficient sequential learning algorithm for growing and pruning rbf (gap-rbf) networks. *IEEE Transactions on Systems Man and Cybernetics Part B-cybernetics*, 34(6):2284–2292, December 2004. [1]
- [4] Guang-Bin Huang. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE Transactions on Neural Networks*, 14(2):274–281, 2003. [1]
- [5] Rui Zhang, Yuan Lan, Guang-Bin Huang, and Zong-Ben Xu. Universal approximation of extreme learning machine with adaptive growth of hidden nodes. *IEEE Transactions on Neural Networks and Learning Systems*, 23(2):365–371, February 2012. [1]
- [6] B. Igel'nik and Y. H. Pao. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 6(6):1320–9, 1995. [1]
- [7] Y. H. PAO, G. H. PARK, and D. J. SOBAJIC. Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing*, 6(2):163–180, April 1994. [1]
- [8] D.S.Broomhead and D.Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988. [1]
- [9] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine. In *Technical Report ICIS/03/2004 (also in <http://www.ntu.edu.sg/ee/icis/cv/egbh Huang.htm>)*, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, January 2004. [1]
- [10] Ming-Bin Li, Guang-Bin Huang, P. Saratchandran, and Narasimhan Sundararajan. Fully complex extreme learning machine. *Neurocomputing*, 68:306–314, 2005. [1]
- [11] Guang-Bin Huang and Chee-Kheong Siew. Extreme learning machine with randomly assigned RBF kernels. *International Journal of Information Technology*, 11(1):16–24, 2005. [1]
- [12] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Real-time learning capability of neural networks. In *Technical Report ICIS/45/2003*, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, April 2003. [1]
- [13] G. B. Huang, H. M. Zhou, X. J. Ding, and R. Zhang. Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems Man and Cybernetics Part B-cybernetics*, 42(2):513–529, April 2012. [1, 6]
- [14] Y. M. Yang, Y. N. Wang, and X. F. Yuan. Parallel chaos search based incremental extreme learning machine. *Neural Processing Letters*, 37(3):277–301, June 2013. [1, 2]
- [15] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 42(1):80–86, February 2000. [3]
- [16] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70:489–501, 2006. [3, 6]
- [17] C. W. Hsu and C. J. Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, March 2002. [4]
- [18] Guorui Feng, Guang-Bin Huang, Qingping Lin, and Robert Gay. Error minimized extreme learning machine with growth of hidden nodes and incremental learning. *IEEE Transactions on Neural Networks*, 20(8):1352–1357, 2009. [6]